1、计算机基础

1.1 语言

- 自然语言--->人与人交流的语言

```
中文、英语、法语、俄语等
```

- 编程语言--->人与机器交流

```
C语言、Java、Javascipt、C#、C++等
```

- 协议--->机器与机器交流

```
HTTP、TCP/IP、UDP、DNS
```

1.2 通信协议

终端设备通过各种协议进行通信，其中互联网协议是互联网的核心。

运行-Mstsc-对服务器进行远程操作

体系结构：

B/S：使用H5技术开发网页应用部署到服务器上，用户在浏览器上访问，这种体系结构就是B/S(Browser/Server)结构。

C/S：使用H5技术开发APP、小程序等应用，服务器端自然是放到服务器上，这种体系结构就是C/S(Client/Server)结构。

通信协议：

- **实体层：**

网卡、网线、WIFI；网卡都有一个独一的编号，mac地址，网卡的mac都是唯一的；

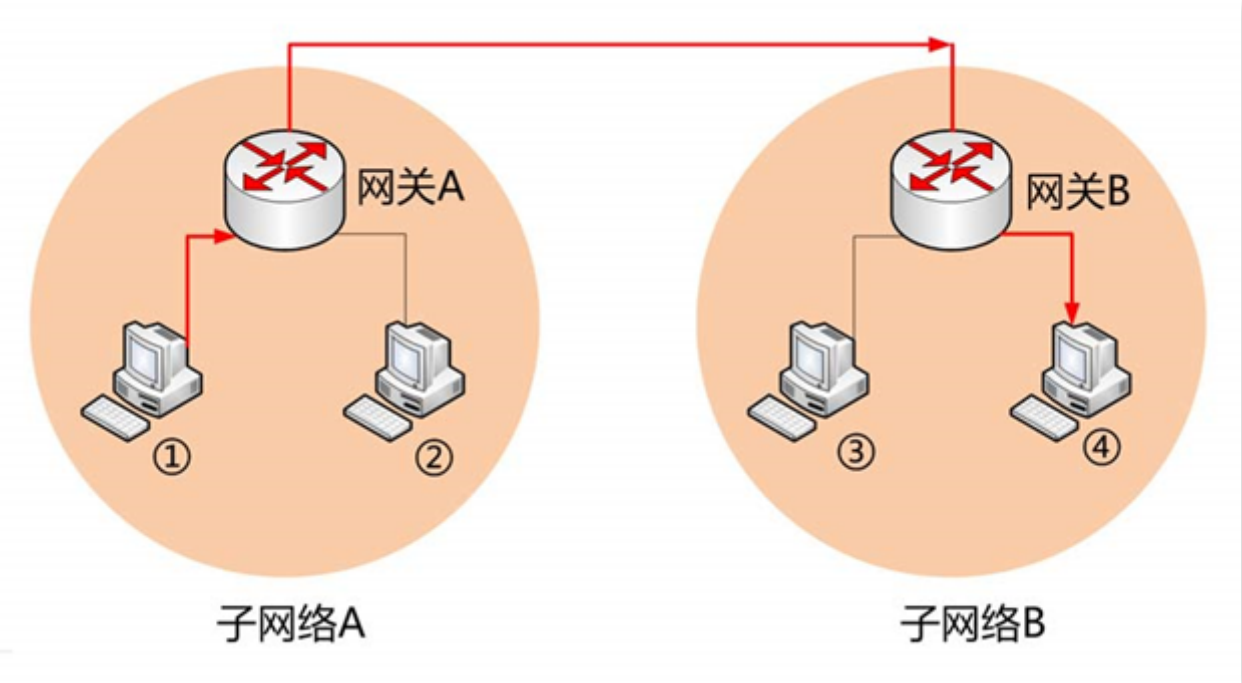
> **终端输入 ipconfig -all 可以得到地址信息**

- **链接层**

对数据进行打包，组成数据包；通过广播的方式在局域网内部进行传输，且只能在局域网内部进行数据传输；广域网；

- **网络层**

IPV4、IPV6；服务器是通过IP地址来找的，不容易记，所以用域名；域名解析：也就是把对应的ip地址解析出来；通过IP地址可以实现局域网与局域网之间的通信，剩余的通信由网关解决。



- **传输层**

端口：是网卡分配给对应程序的编号

HTTP协议：默认是8080

MySQL：3306

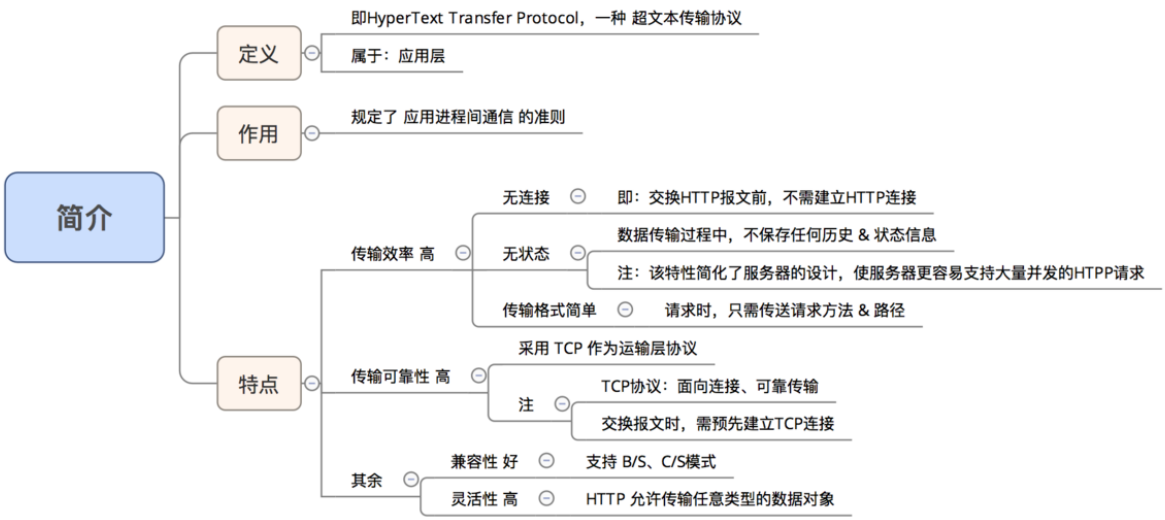
HTTPS：443

FTP：21

- **应用层**

2. HTTP协议

HTTP：HyperTextTransferProtocol，超文本传输协议；



2.1 只能由客户端主动发起请求

a）所有的资源（文件）都需要发起依次请求

i. 所有的资源只是从服务器加载到本地；

ii. 然后在客户端执行；

iii. 在执行的过程之中，如果需要加载资源就继续发起网络请求；

b) 一次请求的四个步骤：

i. 建立连接：open；

ii. 客户端向服务器端发起请求：send request

iii. 服务器端响应数据到客户端：response

iv. 断开链接：close

c) 基于此，我们应该如何优化我们的前端代码加载速度？

i. Script标签放在最后：是阻塞的；

ii. 精灵图：把多个小图标合并到一个图片里面，减少网络请求；

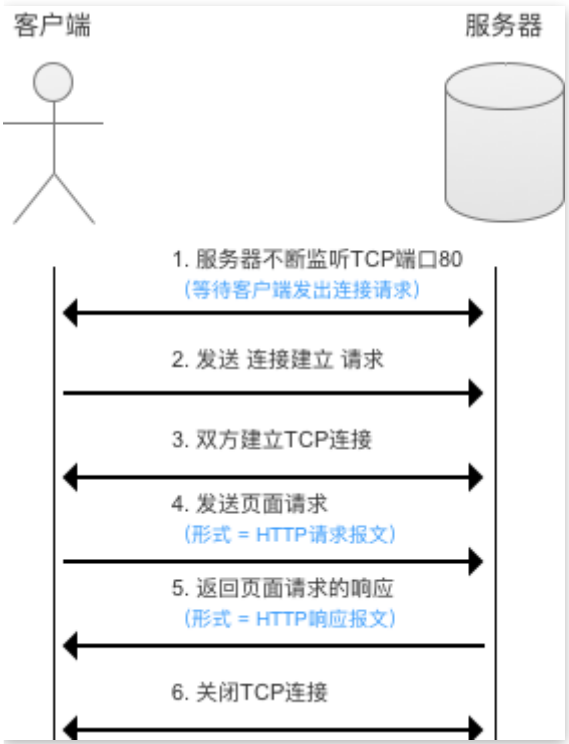
iii. 字体库：把多小小图标合并成一个文件；

iv. 懒加载：把加载时间分散开，提升用户体验，减轻服务器压力；

v. 对JS文件和CSS进行合并压缩；

vi. 只加载需要刷新的数据：AJAX；

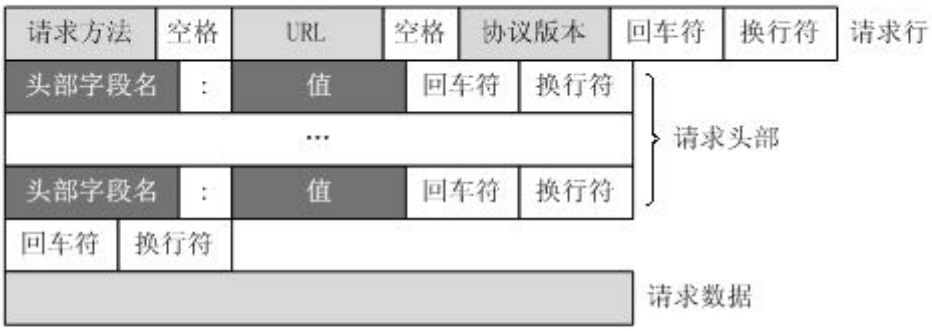
d) 请求的一些关联知识点



2.2 HTTP的请求报文

🌎2.1 **HTTP的请求报文由请求行、请求头、请求体组成：**

报文结构



具体介绍

- 请求行（request line）：声明 请求方法 、主机域名、资源路径 & 协议版本
- 请求头（header）：声明 客户端、服务器 / 报文的部分信息
- 请求体：存放 需发送的数据信息

🌍2.2 请求行：

作用：声明 请求方式、主机域名、资源路径和协议版本

结构：请求方式+请求路径+协议版本

类型	作用	具体介绍	备注
请求方法	定义 对请求对象的操作	• 请求方法有8种：GET、POST、HEAD、DELETE、PUT、DETELTE、TRACE、CONNECT、OPTION • 最常用的是 GET、POST、HEAD • 若服务器 = RESTful接口，则一般会用到GET、POST、DELETE、PUT	• GET　请求读取“URL标志的信息“的信息 • POST　为服务器添加信息 • HEAD　请求读取”URL标志信息的首部“信息 • PUT　为指定的URL下添加（存储）一个文档 • DELETE　删除指定URL所标志的信息 • TRACE　用于进行环回测试的请求报文 • CONNECT　用于代理服务器 • OPTION　请求“选项”的信息
请求路径	URL中的请求地址部分	• 若URL = http://www.baidu.com/，则请求路径 = / • 若URL = http://www.weibo.com/288/home，则请求路径 = /288/home	URL的简介 • 定义：Uniform Resoure Locator，统一资源定位符，一种自愿位置的抽象唯一识别方法 • 作用：表示资源位置 & 访问资源的方法 • 组成：<协议>：//<主机>：<端口>/<路径> a. 协议：应用层通信协议，如在HTTP协议下则是：HTTP：//<主机>：<端口>.. b. 主机：请求资源所在主机的域名 c. 端口 & 路径有时可省略（HTTP默认端口号 = 80）
协议版本	定义HTTP的版本号	常用版本：HTTP/1.0、HTTP/1.1、HTTP/2.0	

🌍2.3 get与post方法的区别：

使用方式	区别			
	传递参数的长度限制	传递参数的数据类型	安全性	应用场景
GET	传递参数长度受限制 • GET 发送的数据存放在URL中：直接附加在URL后、利用1个问号（“?”）代表URL的结尾 & 请求参数的开始 • 而URL 的长度是受限制的（最长度 = 2048 个字符）	只允许 ASCII 字符	差 • 因数据直接添加到URL = 可见 • 浏览器中	• 小量、数据不敏感 • 具体：从指定资源请求数据
POST	不受限制	任何类型	好 • 求参数封装在HTTP请求数据中 • 浏览器中无缓存	• 大量、数据敏感 • 具体：向指定资源提交数据

🌍2.4 请求头：

作用：声明 客户端、服务器 / 报文的部分信息

使用方式：采用**” header（字段名）：value（值） “**的方式

常用请求头

1. 请求和响应报文的通用Header

名称	作用
Content-Type	请求体/响应体的类型，如：text/plain、application/json
Accept	说明接收的类型，可以多个值，用`,`(半角逗号)分开
Content-Length	请求体/响应体的长度，单位字节
Content-Encoding	请求体/响应体的编码格式，如gzip,deflate
Accept-Encoding	告知对方我方接受的Content-Encoding
ETag	给当前资源的标识，和`Last-Modified`、`If-None-Match`、`If-Modified-Since`配合，用于缓存控制
Cache-Control	取值为一般为`no-cache`或`max-age=XX`，XX为个整数，表示该资源缓存有效期(秒)

2. 常见请求Header

名称	作用
Authorization	用于设置身份认证信息
User-Agent	用户标识，如：OS和浏览器的类型和版本
If-Modified-Since	值为上一次服务器返回的`Last-Modified`值，用于确认某个资源是否被更改过，没有更改过(304)就从缓存中读取
If-None-Match	值为上一次服务器返回的 ETag 值，一般会和`If-Modified-Since`一起出现
Cookie	已有的Cookie
Referer	表示请求引用自哪个地址，比如你从页面A跳转到页面B时，值为页面A的地址
Host	请求的主机和端口号

🌎2.5 **请求体：**

作用：存放 需发送给服务器的数据信息

可选部分，如 GET请求就无请求数据

使用方式：共3种

使用方式	说明	实例

数据交换	• 请求体可任意类型
	• 但服务器需额外解析
		{
		"name":"html",
		"year":"5"
		}

键值对	• 键与值之间 用 "=" 连接 • 每个键值对间 用 "&" 连接	key1=value1&key2&value2
分部分形式	请求体被分为多个部分 • 每段以-- {boundary}开头 = 描述头 • 描述头后空一行 接 内容 • 每段以-- {boundary}--结束	（请求体1）-- {boundary} （开头） Content-Disposition：form-data;name="name"（描述头） （空格） I love Carson_Ho （内容） （请求体2）-- {boundary} （开头） Content-Disposition：form-data;name="name"（描述头） （空格） I hate Carson_Ho （内容） （请求体结束标志）-- {boundary} --

2.3 HTTP响应报文

HTTP的响应报文包括：状态行、响应头 & 响应体

🌎3.1 状态行

作用：声明 协议版本，状态码，状态码描述

组成：状态行有协议版本、状态码 &状态信息组成

类型	作用	具体介绍	备注
协议版本	表示 服务器HTTP协议的版本	常用版本：HTTP/1.0、HTTP/1.1、HTTP/2.0	
状态码	表示 服务器返回的响应状态代码	3位十进制数字组成、分为5大类： • 1xx：表示信息通知，如请求收到了或正在进行处理 • 2xx：表示成功，如接受或知道了 • 3xx：表示重定向，如要完成请求还必须采取进一步行动 • 4xx：表示客户端错误，请求包含语法错误 / 无法实现 • 5xx：表示服务器错误，服务器不能实现一种明显无效的请求	• 200：请求成功，请求内容与该响应一起返回 • 202：请求已被接受，但还没处理。 • 301：请求的资源已被永久移动到新的位置。 • 302：请求的资源被临时移动到新的位置。 • 400：请求参数有误，当前请求无法被服务器理解。 • 401：请求需要验证用户 • 403：不允许访问该地址 • 404：Not Found • 408：请求超时 • 500：服务器内部错误 • 502：Bad Gateway网关出错
状态信息	对状态码的简单解释		

图片问题：

```
</meta name="referrer" content="no-referrer">
```

状态行 示例：

HTTP/1.1 202 Accepted(接受)、HTTP/1.1 404 Not Found(找不到)

🌎3.2 响应头

作用：声明客户端、服务器 / 报文的部分信息

使用方式：采用**" header（字段名）：value（值） "**的方式

常用请求头：

1. 请求和响应报文的通用Header

名称	作用
Content-Type	请求体/响应体的类型，如：text/plain、application/json
Accept	说明接收的类型，可以多个值，用`,`(半角逗号)分开
Content-Length	请求体/响应体的长度，单位字节
Content-Encoding	请求体/响应体的编码格式，如gzip,deflate
Accept-Encoding	告知对方我方接受的Content-Encoding
ETag	给当前资源的标识，和`Last-Modified`、`If-None-Match`、`If-Modified-Since`配合，用于缓存控制
Cache-Control	取值为一般为`no-cache`或`max-age=XX`，XX为个整数，表示该资源缓存有效期(秒)

2. 常见响应Header

名称	作用
Date	服务器的日期
Last-Modified	该资源最后被修改时间
Transfer-Encoding	取值为一般为chunked，出现在Content-Length不能确定的情况下，表示服务器不知道响应版体的数据大小，一般同时还会出现`Content-Encoding`响应头
Set-Cookie	设置Cookie
Location	重定向到另一个URL，如输入浏览器就输入baidu.com回车，会自动跳到 https://www.baidu.com ，就是通过这个响应头控制的
Server	后台服务器

3.3 响应体

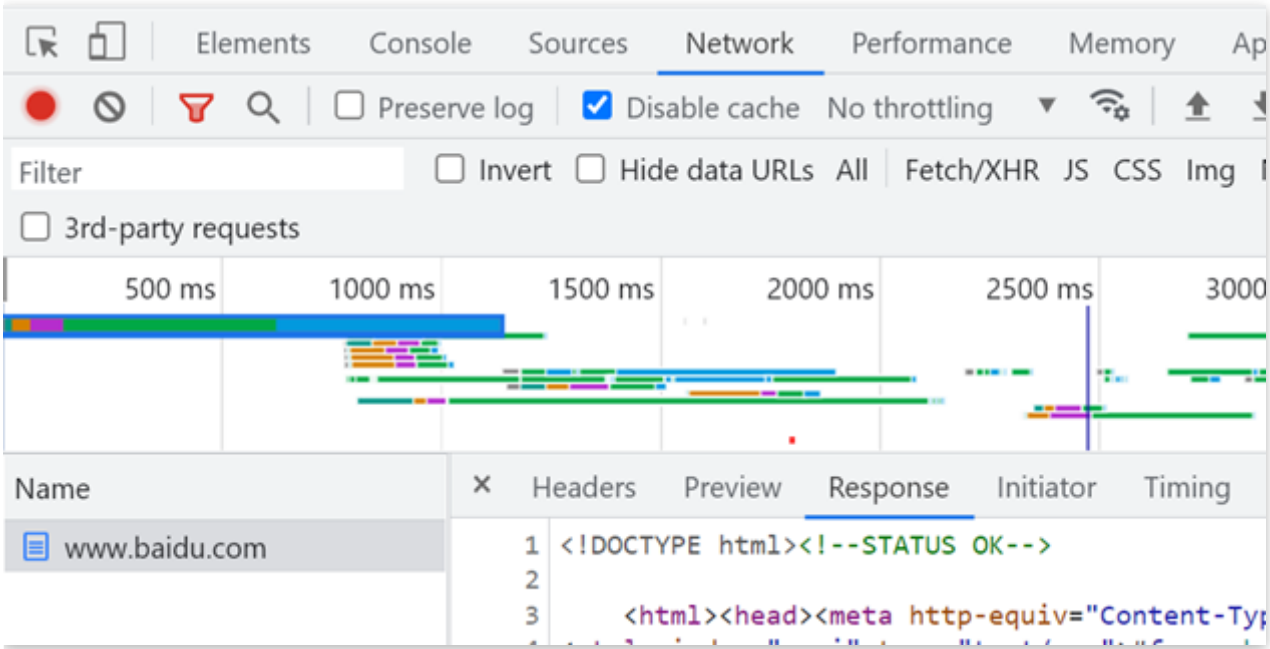
作用：存放需返回给客户端的数据信息

使用方式：和请求体是一致的，同样分为：任意类型的数据交换格式、键值对形式和分部分形式

使用方式	说明	实例

数据交换	• 请求体可任意类型
	• 但服务器需额外解析
		{
		"name":"html",
		"year":"5"
		}

键值对	• 键与值之间 用 "=" 连接 • 每个键值对间 用 "&" 连接	key1=value1&key2&value2
分部分形式	请求体被分为多个部分 • 每段以-- {boundary}开头 = 描述头 • 描述头后空一行 接 内容 • 每段以-- {boundary}--结束	（请求体1）-- {boundary} （开头） Content-Disposition：form-data;name="name"（描述头） （空格） I love Carson_Ho （内容） （请求体2）-- {boundary} （开头） Content-Disposition：form-data;name="name"（描述头） （空格） I hate Carson_Ho （内容） （请求体结束标志）-- {boundary} --



3.4 HTTP 与HTTPS的区别：

HTTPS：HTTP + SSL（SecureSocketLayer ）

i. 数据是需要在网络上进行传递的；

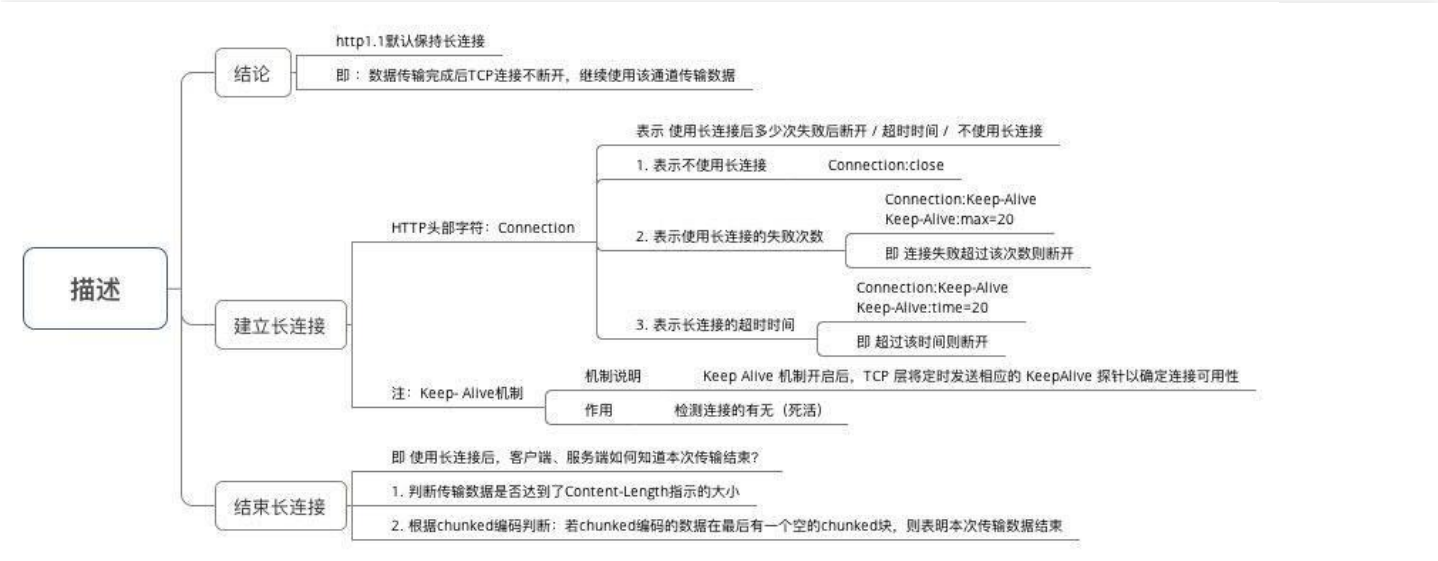
ii. 如果直接暴露一些敏感信息（支付）在网络里面是很不安全的，很容易被嗅探到；

iii. 传输成本会增高，性能较低：加密---解密---证书

iv. 默认端口号443

类型	原理	功能 （数据加密）	性能 （安全性）	使用		
				标准端口	CA申请证书	URL开头
HTTP	应用层	不加密 （明文传输）	不安全	80	不需	http://
HTTPS	传输层	加密 （SSL加密、身份认证）	安全	443	需要	https://

3.5 HTTP处理长连接的方式：



参考原文链接：https://blog.csdn.net/carson_ho/article/details/82106781

2.4HTTP1.0、HTTP1.1、HTTP2.0的区别

HTTP1.0与HTTP1.1的区别：

HTTP1.0:

- 无状态：服务器不跟踪不记录请求过的状态
- 无连接：浏览器每次请求都要建立TCP连接
- 无法复用连接。每次发送请求都需要建立一次TCP连接，而TCP的连接释放过程是比较费事的，导致网络利用率非常低
- 队头堵塞。由于HTTP1.0规定下一个请求必须在前一个响应到达之前才能发送。假设一个请求响应一直不到达，那么下一个请求就不发送，就导致阻塞后面的请求。

HTTP1.1:

- 长连接（connection字段，设置keep-alive可以保持连接不断开）
- 节约带宽
- HOST域
- 缓存处理
- 错误通知管理

HTTP1.1和HTTP2.0的区别：

- **多路复用：**HTTP2.0使用了多路复用的技术，做到同一个连接并发处理多个请求，而且并发请求的数量比HTTP1.1大了好几个数量级。（二进制分帧）
- **头部数据压缩：** HTTP1.1不支持header数据的压缩，HTTP2.0使用HPACK算法对header的数据进行压缩
- **服务器推送：**在客户端请求之前发送数据的机制

3. cookie与session： （无状态）

3.1 Cookie：浏览器用来保存用户信息的文件，可以保存比如用户是谁，购物车有哪些商品等

- 来自于服务器；
- 存储在客户端；
- 安全性得不到保障，不适合存储敏感信息，比如充值、密码等；
- 信息存储量有大小限制：4k；
- 当再次向服务器发起请求的时候，会自动带上cookie；
- cookie 具有不可跨域名性

3.2 SESSION：一次会话，会话是指我们访问网站的一个周期

session在计算机网络应用中被称为“会话控制”。不同的是Cookie保存在客户端浏览器中，而Session保存在服务器上。在服务端保存Session的方法很多，内存、数据库、文件都有。

- 存储在服务器上；所有用户的session保存在服务器上
- SESSION依赖于Cookie；创建session的时候，会自动创建依赖的cookie并响应到客户端，再次发起请求时会自动带上该cookie信息，服务器端的编程语言会自动获取该cookle并根据该cookie找到对应的session,我们写代码的时候直接用这个session，不用管依赖的cookie的；
- 安全性比较高，可以存储敏感信息；
- 没有大小限制；

1. 客户端浏览器访问网站的时候,
2. 服务器会向客户浏览器发送一个每个用户特有的会话编号sessionID，让浏览器写入到cookie里(大多数情况)。服务器同时也把sessionID和对应的用户信息、用户操作记录在服务器上，这些记录就是session。
3. 客户端浏览器再次访问时，会发送cookie给服务器，cookie中就包含sessionID。
4. 服务器从cookie里找到sessionID，再根据sessionID找到以前记录的用户信息就可以知道他是谁, 之前操控哪些、访问过哪里。

3.3 基于Token的身份验证的过程如下（token ——令牌）

Acesss Token:

> 访问资源接口（API）时所需要的资源凭证
> 简单 token 的组成：uid(用户唯一的身份标识)、time(当前时间的时间戳)、sign（签名，token 的前几位以哈希算法压缩成的一定长度的十六进制字符串）

1. 用户通过用户名和密码发送请求。
2. 服务端验证, 返回生成的token 给客户端, 同时给数据库和Redis里关联token和用户信息。
3. 客户端储存token,并且其后的每一次请求都添加token, token应该在HTTP的头部发送从而保证了Http请求无状态。
4. 服务端查询Redis+数据库, 验证token并返回数据。
5. token的优势：

- 无状态、可扩展

- 支持移动设备
- 跨服务器调用
- 安全

6. token 的身份验证流程：

- 客户端使用用户名跟密码请求登录
- 服务端收到请求，去验证用户名与密码
- 验证成功后，服务端会签发一个 token 并把这个 token 发送给客户端
- 客户端收到 token 以后，会把它存储起来，比如放在 cookie 里或者 localStorage 里
- 客户端每次向服务端请求资源的时候需要带着服务端签发的 token
- 服务端收到请求，然后去验证客户端请求里面带着的 token ，如果验证成功，就向客户端返回请求的数据
- 每一次请求都需要携带 token，需要把 token 放到 HTTP 的 Header 里
- 基于 token 的用户认证是一种服务端无状态的认证方式，服务端不用存放 token 数据。用解析 token 的计算时间换取 session 的存储空间，
- 从而减轻服务器的压力，减少频繁的查询数据库
- token 完全由应用管理，所以它可以避开同源策略

3.4 cookie、session、token的比较:

cookie :

1. cookie由服务器生成，保存在客户端浏览器。
2. 容易被劫持，不安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗。
3. cookie可以被用户禁止
4. 容量小, 单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

session:

1. session是由应用服务器维持的一个服务器端的存储空间, 没有对存储的数据量的限制，可以保存更为复杂的数据类型.
2. session 默认被存在在服务器的一个文件里, 但是实际中可以放在 文件、数据库、或内存中都可以。
3. 当用户量增多时，会对服务器造成较大压力。
4. Session的实现方式大多数情况用Cookie保存的，但是也可以使用URL地址重写。
5. 较安全,用户验证这种场合一般会用 session, 比如金融银行类的产品,

token:

1.无状态、可扩展
2.支持移动设备
3.跨服务器调用
4.安全

参考原文链接：https://blog.csdn.net/u014600626/article/details/107807029

3.5 JWT:

JSON Web Token（简称 JWT）是目前最流行的跨域认证解决方案。

是一种认证授权机制。

JWT 是为了在网络应用环境间传递声明而执行的一种基于 JSON 的开放标准（RFC 7519）。

JWT 的声明一般被用来在身份提供者 和服务提供者间传递被认证的用户身份信息，以便于从资源服务器获取资源。比如用在用户登录上。

可以使用 HMAC 算法或者是 RSA 的公/私秘钥对 JWT 进行签名。因为数字签名的存在，这些传递的信息是可信的。

1.生成JWT

```
jwt.io/www.jsonwebtoken.io/
```

2.JWT 认证流程

```
用户输入用户名/密码登录，服务端认证成功后，会返回给客户端一个 JWT；
客户端将 token 保存到本地（通常使用 localstorage，也可以使用 cookie）；
当用户希望访问一个受保护的路由或者资源的时候，需要请求头的 Authorization 字段中使用Bearer 模式添加 JWT，其内容看起来是下面这样
```

```
服务端的保护路由将会检查请求头 Authorization 中的 JWT 信息，如果合法，则允许用户的行为
因为 JWT 是自包含的（内部包含了一些会话信息），因此减少了需要查询数据库的需要
因为 JWT 并不使用 Cookie 的，所以你可以使用任何域名提供你的 API 服务而不需要担心跨域资源共享问题（CORS）
因为用户的状态不再存储在服务端的内存中，所以这是一种无状态的认证机制
```

3.JWT 的使用方式

方式一

当用户希望访问一个受保护的路由或者资源的时候，可以把它放在 Cookie 里面自动发送，但是这样不能跨域，所以更好的做法是放在 HTTP 请求头信息的 Authorization 字段里，使用 Bearer 模式添加 JWT。

方式二

跨域的时候，可以把 JWT 放在 POST 请求的数据体里。

方式三

通过 URL 传输：

http://www.example.com/user?token=xxx
项目中使用 JWT

3.6 Token 和 JWT 的区别

相同：

- 都是访问资源的令牌
- 都可以记录用户的信息
- 都是使服务端无状态化
- 都是只有验证成功后，客户端才能访问服务端上受保护的资源

区别：

- Token：服务端验证客户端发送过来的 Token 时，还需要查询数据库获取用户信息，然后验证 Token 是否有效。
- JWT：将 Token 和 Payload 加密后存储于客户端，服务端只需要使用密钥解密进行校验（校验也是 JWT 自己实现的）即可，不需要查询或者减少查询数据库，因为 JWT 自包含了用户信息和加密的数据。

JWT参考原文链接：https://blog.csdn.net/weixin_58045199/article/details/125679498

4. URL、URI

URL：统一资源定位符
URI：统一资源标识符 ID 身份证